Safety Proofs using Appearance and Behaviours

Sumanth Prabhu S, Kumar Madhukar, R Venkatesh

TRDDC, Pune

July 20, 2018

Inductive Invariants

```
int x = y = 0
while (*) {
  x = x + 1
  y = y + x
}
assert(y >= 0)
```

Program Reference: Understanding IC3

Inductive Invariants

```
int x = y = 0
while (*) {
  x = x + 1
  y = y + x
}
assert(y >= 0)
```

Safe Inductive Invariants:

$(x \geq 0 \land y \geq 0)$

$(x \geq 0 \land y - x \geq 0)$

Program Reference: Understanding IC3

Inductive Invariants

Given $\langle V \cup V', \mathit{Init}, \mathit{Tr} \rangle$ and Bad

Initiation: $\mathit{Init}(V) \Rightarrow \mathit{Inv}(V)$

Consecution: $\mathit{Inv}(V) \wedge \mathit{Tr}(V, V') \Rightarrow \mathit{Inv}(V')$

Safety: $\mathit{Inv}(V) \wedge \mathit{Bad}(V) \Rightarrow \mathit{false}$

Inductive Invariants

Given $\langle V \cup V', Init, Tr \rangle$ and Bad
$\{x, y, x', y'\}, x = 0 \land y = 0$, $x' = x + 1 \land y' = y + x$ and $\neg(y \geq 0)$

Initiation: $Init(V) \Rightarrow Inv(V)$
$(x = 0 \land y = 0) \Rightarrow (x \geq 0 \land y \geq 0)$

Consecution: $Inv(V) \land Tr(V, V') \Rightarrow Inv(V')$
$(x \geq 0 \land y \geq 0) \land x' = x + 1 \land y' = y + x \Rightarrow (x' \geq 0 \land y' \geq 0)$

Safety: $Inv(V) \land Bad(V) \Rightarrow false$
$(x \geq 0 \land y \geq 0) \land \neg(y \geq 0) \Rightarrow false$

Inductive Invariants

Given $\langle V \cup V', \mathit{Init}, \mathit{Tr} \rangle$ and Bad
$\{x, y, x', y'\}, x = 0 \land y = 0, x' = x + 1 \land y' = y + x$ and $\neg(y \geq 0)$

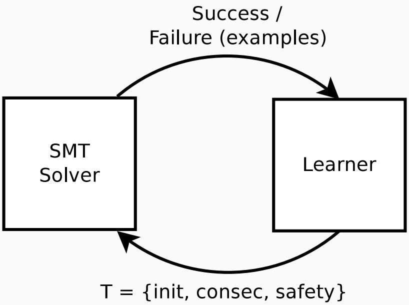
Initiation: $\mathit{Init}(V) \Rightarrow \mathit{Inv}(V)$
$(x = 0 \land y = 0) \Rightarrow (x \geq 0 \land y \geq 0)$

Consecution: $\mathit{Inv}(V) \land \mathit{Tr}(V, V') \Rightarrow \mathit{Inv}(V')$
$(x \geq 0 \land y \geq 0) \land x' = x + 1 \land y' = y + x \Rightarrow (x' \geq 0 \land y' \geq 0)$

Safety: $\mathit{Inv}(V) \land \mathit{Bad}(V) \Rightarrow \mathit{false}$
$(x \geq 0 \land y \geq 0) \land \neg(y \geq 0) \Rightarrow \mathit{false}$

How to synthesize *Inv*?

Guess and Check



Iterative learning: $Inv \Leftrightarrow I_0 \wedge I_1 \wedge \cdots \wedge I_n$

Appearance Guided Synthesis

```
int x = y = 0
while (*)
  x = x + 1
  y = y + x

assert(y >= 0)
```

Probability distribution:

$(x \geq 0) \mapsto 0.4$

$(-x \geq 0) \mapsto 0.0$

$(y \geq 0) \mapsto 0.3$

$(-y \geq 0) \mapsto 0.0$

$(x + y \geq 0) \mapsto 0.2$

$(y - x \geq 0) \mapsto 0.1$

Appearance Guided Synthesis

Fedyukovich, Kaufman, and Bodík, FMCAD 2017

```
int x = y = 0
while (*)
  x = x + 1
  y = y + x

assert(y >= 0)
```

Sampling Grammar

$c ::= 0 \mid 1 \mid -1$

$k ::= 0 \mid 1 \mid -1$

$v ::= x \mid y$

$lincom ::= k \cdot v + \ldots k \cdot v$

$ineq ::= lincom \geq c \mid lincom > c$

$cand ::= ineq \vee ineq \vee \ldots ineq$

Appearance Guided Synthesis

```
int x = y = 0
while (*)
  x = x + 1
  y = y + x

assert(y >= 0)
```

How often does a disjunctive formula have the arity *i*

How often does an operator *op* $\in \{>, \geq\}$ appear among the inequalities

How often does a variable *v* have a coefficient *k*

Appearance Guided Synthesis

```
int x = y = 0
while (*)
  x = x + 1
  y = y + x

assert(y >= 0)
```

How often does a disjunctive formula have the arity *i*

$$p_{\vee}(2) = 0$$

How often does an operator $op \in \{>, \geq\}$ appear among the inequalities

$$p_{>} = 1/5$$

How often does a variable *v* have a coefficient *k*

$$p_{\{1,x\}}(1) = 1/2$$

Detective Auguste Dupin gave them a 'stong acceptance' as they found what was hidden in plain sight.

Relearning Probabilities

Avoid candidates that are:

- Already checked
- Stronger than failures

  $$(x > 5 \vee x + y \geq 0) \supset (x > 10 \vee x + y > 5)$$

- Weaker than learned lemmas

  $$(y \geq 0 \vee y - x \geq 10) \subset (y \geq -1 \vee y - x > 8)$$

Increase probability of candidates that are unrelated

Experimental Evaluation

On 76 loopy programs, this technique outperformed

- ▷ μZ on 37 benchmarks (including 32 for which μZ crashed or timed out after 10 minutes)
- ▷ ICE-DT on 53 benchmarks (including 30 . . .)
- ▷ MCMC on 67 benchmarks (including 49 . . .)

Downsides

Equal treatment of all syntactic expressions

Ignorance to whether the candidates have a semantic value

Inability to predict an appropriate order of candidates to be sampled and checked

Downsides

```
int x = k = c = 0;
int N = *;
while (c < N)
  int M = *;
  if (k mod 2 == 0)
      x = x + M;
  c = c + M;
  k = x + c;

assert(x >= N);
```

Inductive Invariant1:
$k\ mod\ 2 = 0 \land x = c$

Inductive Invariant2:
$k = x + c \land x = c$

Accelerating Synthesis

Fedyukovich, and Bodík, TACAS 2017

Usage of Interpolation

Safety Proofs from Bounded Model Checking

Batch-wise candidate check

for each $\mathit{cand} \in \mathit{candidates}$

$$\bigwedge_{c \in \mathit{candidates}} c(V) \bigwedge \mathit{Tr}(V, V') \Rightarrow \mathit{cand}(V')$$

Accelerating Synthesis

```
int x = k = c = 0;
int N = *;
while (c < N)
  int M = *;
  if (k mod 2 == 0)
      x = x + M;
  c = c + M;
  k = x + c;

assert(x >= N);
```

BMC:
$x = 0 \wedge k = 0 \wedge c = 0 \wedge \neg(c < N) \wedge \neg(x \geq N)$

Accelerating Synthesis

```
int x = k = c = 0;
int N = *;
while (c < N)
  int M = *;
  if (k mod 2 == 0)
      x = x + M;
  c = c + M;
  k = x + c;

assert(x >= N);
```

BMC:
$x = 0 \wedge k = 0 \wedge c = 0 \wedge \neg(c < N) \wedge \neg(x \geq N)$

Interpolants:
$\{x \geq 0, c \leq 0\}, \{x = c\}, \{x \geq c\}$

Accelerating Synthesis

```
int x = k = c = 0;
int N = *;
while (c < N)
  int M = *;
  if (k mod 2 == 0)
      x = x + M;
  c = c + M;
  k = x + c;

assert(x >= N);
```

BMC:
$x = 0 \wedge k = 0 \wedge c = 0 \wedge \neg(c < N) \wedge \neg(x \geq N)$

Interpolants:
$\{x \geq 0, c \leq 0\}, \{x = c\}, \{x \geq c\}$

Candidates:
$k = x + c \wedge k \; mod \; 2 = 0$

Behaviour as Data

Prabhu, Madhukar, Venkatesh, SAS 2018, *to appear*

```
assume(1 <= n <= 1000);
sum = 0, i = 1;
while(i<=n) {
  sum = sum + i;
  i = i + 1;
}
assert(2*sum == n*(n+1));
```

Behaviour as Data

```
assume(1 <= n <= 1000);
sum = 0, i = 1;
while(i<=n) {
  sum = sum + i;
  i = i + 1;
}
assert(2*sum == n*(n+1));
```

Safe Inductive invariant:

$$2 * sum = i * (i - 1) \land i \leq n + 1$$

Behaviour as Data

```
assume(1 <= n <= 1000);
sum = 0, i = 1;
if(i<=n) {                <0 , 1>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <1,  2>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <3,  3>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <6,  4>
  sum = sum + i;
  i = i + 1;
}
```

Behaviour as Data

```
assume(1 <= n <= 1000);
sum = 0, i = 1;
if(i<=n) {                <0 , 1>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <1,  2>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <3,  3>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <6,  4>
  sum = sum + i;
  i = i + 1;
}
```

If an invariant is a conjunction of k polynomial equations each of degree d and nullity of A is k, where A is a data matrix, then any basis for nullspace of A forms an invariant.

Sharma et al, ESOP, 2013

Behaviour as Data

```
assume(1 <= n <= 1000);
sum = 0, i = 1;
if(i<=n) {                <0 , 1>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <1,  2>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <3,  3>
  sum = sum + i;
  i = i + 1;
}
if(i<=n) {                <6,  4>
  sum = sum + i;
  i = i + 1;
}
```

Inductive invariant:

$$a * sum^2 + b * i^2 + c * sum * i + d * sum + e * i + f = 0$$

Algebraic Invariants

1	sum	i	sum^2	$sum * i$	i^2
1	0	1	0	0	1
1	1	2	1	2	4
1	3	3	9	9	9
1	6	4	36	24	16
1	10	5	100	50	25

Algebraic Invariants

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 2 & 1 & 2 & 4 \\ 1 & 3 & 3 & 9 & 9 & 9 \\ 1 & 6 & 4 & 36 & 24 & 16 \\ 1 & 10 & 5 & 100 & 50 & 25 \end{bmatrix} * \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = 0$$

Algebraic Invariants

$$basis(Nullspace(M)) = \begin{bmatrix} 0 \\ -2 \\ -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$0 * 1 - 2 * sum - 1 * i + 0 * sum^2 + 0 * sum * i + 1 * i^2$$

$$2 * sum = i * (i - 1)$$

Conditional Invariants

```
int LRG = nondet();
assume(LRG > 0);

int x = 0, y = LRG;

while(x < 2*LRG) {
  if (x < LRG) {
    y = y;
  } else {
    y = y + 1;
  }
  x = x + 1;
}
assert(y == 2*LRG);
```

Conditional Invariants

```
int LRG = nondet();
assume(LRG > 0);

int x = 0, y = LRG;

while(x < 2*LRG) {
  if (x < LRG) {
    y = y;
  } else {
    y = y + 1;
  }
  x = x + 1;
}
assert(y == 2*LRG);
```

Disjunctive:

$((x \geq \mathsf{LRG}) \vee$
$(y = x) \wedge$
$(x \leq 2 * \mathsf{LRG}))$

Conditional Invariants

```
int LRG = nondet();
assume(LRG > 0);

int x = 0, y = LRG;

while(x < 2*LRG) {
  if (x < LRG) {
    y = y;
  } else {
    y = y + 1;
  }
  x = x + 1;
}
assert(y == 2*LRG);
```

Disjunctive:

$((x \geq \mathrm{LRG}) \vee$
$(y = x) \wedge$
$(x \leq 2 * \mathrm{LRG}))$

Conditional Invariants:

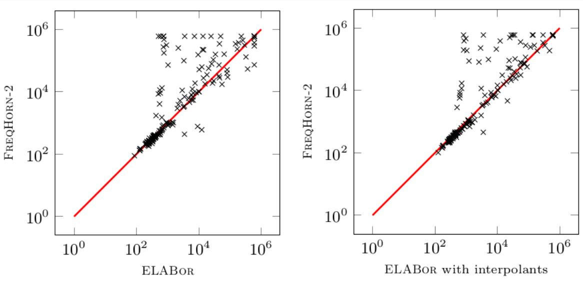
$(((x < \mathrm{LRG}) \Rightarrow (y = \mathrm{LRG})) \wedge$
$((x \geq \mathrm{LRG}) \Rightarrow (y = x))) \wedge$
$(x \leq 2 * \mathrm{LRG})$

CTIs: $s_k \models Inv$ and $(s_k, s'_{k+1}) \models Tr$, but $s'_{k+1} \not\models Inv'$

Results

ELABoR solved 16/24 new programs when compared to FreqHorn-2

2x speedup and 100s time difference for 31 programs



Future Directions

Usage in solving nested loops (*In Review*)

Neural nets to refine sampling

Deciding between behaviour or appearance

Conclusion

Appearance guided synthesis

Behaviours to obtain candidates

Conditional invariants for disjunctions

References

Grigory Fedyukovich and Rastislav Bodík. "Accelerating Syntax-Guided Invariant Synthesis". In: *2018 Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2018, Thessaloniki, Greece, April 14-20, 2018. To appear*.

Grigory Fedyukovich, Samuel J. Kaufman, and Rastislav Bodík. "Sampling invariants from frequency distributions". In: *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*. 2017, pp. 100–107.

Sumanth Prabhu S, Kumar Madhukar, and R Venkatesh. "Efficiently Learning Safety Proofs from Appearance as well as Behaviours". In: *Static Analysis Symposium*. 2018, to appear.

Rahul Sharma et al. "A Data Driven Approach for Algebraic Loop Invariants". In: *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*. 2013, pp. 574–592.

Algorithm 5 ELABOR: Learning from Behaviours and CTIs

Input: *Init*, *Tr*, *Bad* and V
Output: *lemmas*

$behaviours \leftarrow$ EXECUTE($Init$, Tr, Bad)
$candidates \leftarrow$ GETALGEBRAICCANDIDATES($behaviours$)
$\mathcal{P} \leftarrow$ COMPUTEDISTRIBUTION($Init$, Tr, Bad)
$G \leftarrow$ CONSTRUCTGRAMMAR($\mathcal{P}$)
$\mathcal{P}_a \leftarrow$ COMPUTEDISTRIBUTION(Tr_{conds})
$G_a \leftarrow$ CONSTRUCTGRAMMAR($\mathcal{P}_a$)
$L \leftarrow \emptyset$ ▷ the set of lemmas
$disjunct \leftarrow false$
while $\bigwedge_{l \in L} l(V) \wedge Bad(V)$ is SAT **do**
  if $\neg disjunct$ **then** $disjunct \leftarrow$ CHECKFORIMPL($CTIs$)
  if $disjunct$ **then** $antecedent \leftarrow$ NEWCANDIDATE(G_a)
  while $|candidates| < BatchSize$ **do** ▷ for a pre-decided $BatchSize$
    $cand \leftarrow$ NEWCANDIDATE(G)
    if $init \leftarrow Init(V) \wedge \neg cand(V)$ is UNSAT **then**
      if $disjunct$ **then** $candidates \leftarrow candidates \cup \{antecedent \Rightarrow cand\}$
      else $candidates \leftarrow candidates \cup \{cand\}$
    else ADJUST($cand, G, \mathcal{P}$)
  for $cand \in candidates$ **do**
    if $consec \leftarrow \bigwedge_{c \in candidates} c(V) \bigwedge_{l \in L} l(V) \wedge Tr(V, V') \wedge \neg cand(V')$ is SAT **then**
      $candidates \leftarrow candidates \setminus \{cand\}$
      ADJUST($cand, G, \mathcal{P}$)
      $CTIs \leftarrow CTIs \cup \{$GETMODEL$(V)\} \cup \{$GETMODEL$(V')\}$
      $candidates.reset$ ▷ start the loop afresh
  if $disjunct \wedge |candidates| > 0$ **then** ADJUST($antecedent, G_a, \mathcal{P}_a$)
  for $cand \in candidates$ **do** $L \leftarrow L \cup \{cand\}$
return L

Conditional Invariants

Polynomial relation: $x_i' \in V'$,
$f(x_i') =$
$c_1 * m_1 + c_2 * m_2 + \cdots + c_n * m_n$

Conditional Invariants

Polynomial relation: $x'_i \in V'$,
$f(x'_i) =$
$c_1 * m_1 + c_2 * m_2 + \cdots + c_n * m_n$

CTIs: $s_k \models \mathit{Inv}$ and $(s_k, s'_{k+1}) \models \mathit{Tr}$,
but $s'_{k+1} \not\models \mathit{Inv}'$

Conditional Invariants

Polynomial relation: $x'_i \in V'$,
$f(x'_i) =$
$c_1 * m_1 + c_2 * m_2 + \cdots + c_n * m_n$

CTIs: $s_k \models Inv$ and $(s_k, s'_{k+1}) \models Tr$,
but $s'_{k+1} \not\models Inv'$

$\mathbf{M}$ matrix of monomials from s_k
$\vec{f}_{x'_i}^{\,T} = \begin{pmatrix} x'_{i_1} & \ldots & x'_{i_l} \end{pmatrix}$ from s'_{k+1}

Conditional Invariants

Polynomial relation: $x'_i \in V'$,
$f(x'_i) =$
$c_1 * m_1 + c_2 * m_2 + \cdots + c_n * m_n$

CTIs: $s_k \models Inv$ and $(s_k, s'_{k+1}) \models Tr$,
but $s'_{k+1} \not\models Inv'$

$\mathbf{M}$ matrix of monomials from s_k
$\vec{f_{x'_i}}^T = \begin{pmatrix} x'_{i_1} & \ldots & x'_{i_l} \end{pmatrix}$ from s'_{k+1}

$rank(\mathbf{M}) \neq rank((\mathbf{M}|\vec{f_{x'_i}}))$ no
solution over $c_1 \ldots c_n$

Conditional Invariants

```
int LRG = nondet();
assume(LRG > 0);

int x = 0, y = LRG;

while(x < 2*LRG) {
  if (x < LRG) {
    y = y;
  } else {
    y = y + 1;
  }
  x = x + 1;
}
assert(y == 2*LRG);
```

$y = LRG$

$s_k(LRG) = 100$, $s_k(x) = 100$,
$s_k(y) = 100$

$s'_{k+1}(LRG') = 100$,
$s_{k+1}(x') = 101$,
$s_{k+1}(y') = 101$

Conditional Invariants

```
int LRG = nondet();
assume(LRG > 0);

int x = 0, y = LRG;

while(x < 2*LRG) {
  if (x < LRG) {
    y = y;
  } else {
    y = y + 1;
  }
  x = x + 1;
}
assert(y == 2*LRG);
```

$y = LRG$

$s_k(LRG) = 100, s_k(x) = 100,$
$s_k(y) = 100$

$s'_{k+1}(LRG') = 100,$
$s_{k+1}(x') = 101,$
$s_{k+1}(y') = 101$

$y = x$

$s_k(LRG) = 100, s_k(x) = 10,$
$s_k(y) = 100$

$s'_{k+1}(LRG') = 100,$
$s_{k+1}(x') = 11,$
$s_{k+1}(y') = 100$

Conditional Invariants

```
int LRG = nondet();
assume(LRG > 0);

int x = 0, y = LRG;

while(x < 2*LRG) {
  if (x < LRG) {
    y = y;
  } else {
    y = y + 1;
  }
  x = x + 1;
}
assert(y == 2*LRG);
```

$$\mathit{Rank}\begin{bmatrix} \mathit{const} & \mathit{LRG} & x & y \\ 1 & 100 & 100 & 100 \\ 1 & 100 & 10 & 100 \\ \ldots & & & \end{bmatrix}$$

$$\neq$$

$$\mathit{Rank}\begin{bmatrix} \mathit{const} & \mathit{LRG} & x & y & y' \\ 1 & 100 & 100 & 100 & 101 \\ 1 & 100 & 10 & 100 & 100 \\ \ldots & & & & \end{bmatrix}$$

The cardinality of **B** is called *dimension* of **V**. For a matrix **A**, the dimension of the vector space generated by its columns is called its *rank*. The *nullspace* of a matrix **A** is a set of all vectors **v** such that **Av** $= 0$. The dimension of a matrix's nullspace is also called its *nullity*.